Leipzig

Week 4-5 | Coursework 1

Set by: Amna Asif Total marks 100

Submit by: 23:59, 18th February 2024 (via Moodle)

This is an individual assignment that will count for 30% of your overall marks for this module. The marks will be based on a) the correctness of the solution and b) the corresponding explanation of how the solution was achieved.

Learning objectives

- Develop appreciation and understanding of security tools.
- Formulate troubleshooting methods to identify/solve problems.
- Evaluate information to critically argument solution choices.
- Effectively communicate ideas.

Submission requirements

- Properly justify and explain your solutions. This may include pseudocode and concise comments to demonstrate your understanding. Please use a text editor to prepare this. The submission should be in 2 different file formats that a) include your explanation for all 3 tasks and must be uploaded on Moodle as a single PDF file and b) your executable code.
- Provide extra documentation in support of the solution you provided. In this document, you are required to explain every detail of the solution e.g. how you perform matrix multiplication in your exercise 1 solution? The same applies to all 3 exercises.
- Please upload your complete solution to Moodle by the deadline.
- Your code and documentation must not be copied from other sources. Provide proper citations and references for all the resources you are using to seek help in solving these exercises.

Marking guidelines

Task 1: Weight 30% of total marks

- 50% of these marks will be allocated based on the correctness of your solution.
- 50% of these marks will be allocated based on the submitted explanation of your solution.

Task 2: Weight 30% of total marks

- 50% of these marks will be allocated based on the correctness of your solution.
- 50% of these marks will be allocated based on the submitted explanation of your solution.

Task 3: Weight 30% of total marks

- 50% of these marks will be allocated based on the correctness of your solution.
- 50% of these marks will be allocated based on the submitted explanation of your solution.

Task 4: Weight 10% of total marks

This component is assigned to the correctness and completeness of your code and report. You need to follow the checklist below:

- Is your report well-structured and well-presented? (including sections and naming conventions)
- Is your code well-structured and documented?
- Have you read the coursework assessment requirements correctly?
- Have you tried to follow all submission requirements?
- Have you adhered to all the rules related to this assignment?
- Are there proper references and citations?
- All the supplementary files are provided with the submission.
- There are clear instructions on how to run a code.

Plagiarism policy:

https://www.lancaster.ac.uk/media/lancaster-university/content-assets/documents/student-based-services/asq/principles-policies-and-guidelines/Plagiarism-Framework.pdf

https://portal.lancaster.ac.uk/ask/study/developing-academic-skills/using-ai-in-your-learning-and-assessment/

Exercise 1: Implementation of Hill Cipher

Implement the encryption function for a Hill cipher. The Hill cipher is a polygraphic substitution cipher based on linear algebra. In this cipher, the text is divided into block matrices, and each block is encrypted using a matrix multiplication operation. To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption. Your implementation should support only uppercase ASCII letters, and any character outside this set should remain unchanged during encryption.

Reading: https://en.wikipedia.org/wiki/Hill_cipher

Define a function encrypt that takes parameters: plaintext and key matrix, and a function decrypt that takes cipher text and key matrix. The first prompt should ask the user to enter the plain text for encryption. The system code must present both output from:

plaintext-> ciphertext & ciphertext-> plaintext.

Example code

```
def encrypt(plain_text,key_matrix):
# Your code goes here
def decrypt(cipher_text,key_matrix):
# Your code goes here
if __name__ == "__main__":
# Your code goes here
print("cipher text: ", encrypt(plain_text, key_matrix)
print("plain text: ", decrypt(cipher_text, key_matrix)
# Please note you can add and modify functions, variables and parameters
according to your code requirements
# Example input and output:
plaintext entered= "TEST"
ciphertext= "OSEJ"
plaintext="TEST"
```

Exercise2: The Diffie Hellman Man-in-the-Middle Attack

The Diffie Hellman key exchange protocol is an asymmetric scheme. The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent symmetric encryption of messages.
Required reading: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange
You must implement and demonstrate a man-in-the-middle attack on the Diffie Hellman key exchange protocol. According to the scheme, Alice and Bob want to use the Diffie Hellman key exchange to calculate the secure key. However, Darth is the opponent who attacks their key exchange and takes over the communication on secure key calculation.

1. Darth prepares for the attack by generating two random private keys XD_1 and XD_2 and then computing the corresponding public keys YD_1 and YD_2.
2. Alice transmits YA to Bob.
3. Darth intercepts YA and transmits YD_1 to Bob. Darth also calculates K_2 = (YA) XD_2 mod q.
4. Bob receives YD_1 and calculates K_1 = ($YD1$) XB mod q.
5. Bob transmits YB to Alice.
6. Darth intercepts YB and transmits YD_2 to Alice. Darth calculates K_1 = (YB) XD_1 mod q.
7. Alice receives YD_2 and calculates K_2 = (YD_2) XA mod q.
At this point, Bob and Alice think that they share a secret key, but instead, Bob and Darth share secret key K_1, and Alice and Darth share secret key K_2. All future communication between Bob and Alice is compromised in the following way.
1. Alice sends an encrypted message M: E (K_2, M).
2. Darth intercepts the encrypted message and decrypts it to recover M.
3. Darth sends Bob E (K_1, M) or E (K_1, M=), where M= is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob.

Reference: Stallings, William. "CRYPTOGRAPHY AND NETWORKSECURITY PRINCIPLES ANDPRACTICE." (2011).

Use the Diffie-Hellman key exchange protocol to exchange a secret between User1 and User2. Implement the interruption of User3, who is the attacker.

Your implementation should use the HMAC-based Extract-and-Expand key derivation function with the following setup:

- SHA256 for the algorithm
- length = 32
- salt=None and
- info=b'handshake data'.

Define a function Diffie_Hellman() that takes no parameters and returns both your public key (in PEM format) and the calculated derived key in bytes.

Reading: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange
https://cryptography.io/en/latest/hazmat/primitives/asymmetric/dh/

Example code format:

```
# Required modules
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import dh
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.primitives.serialization
  import Encoding, PublicFormat, load_pem_parameters,
  load_pem_public_key, load_pem_private_key

def Diffie_Hellman():
  #TODO
  return #(Your public key in PEM format AND your derived key in bytes)
```

Example test case
Each result is unique. Your public key and calculated derived key will be used to evaluate the correctness of your implementation.
Also, demonstrate the interruption of User3, who is the attacker in the key exchange.
Implement two cases:

1. User 3 (attacker) wants to eavesdrop on the communication without altering it.
2. User 3 (attacker) wants to modify the message going to User2.

Exercise 3: Elliptic Curve Digital Signature Algorithm
Implement a secure communication system for digital signatures using Elliptic Curve Cryptography (ECC). You aim to encode a plain text message and generate a hashed digital signature using elliptic curve key generation. Such that

1. Implements a function to generate an elliptic curve key pair (private key and corresponding public key).
2. Write a function that takes a plain text message as input and produces its SHA-256 hash value.
3. Implement a function to sign the hashed message using the private key generated in step 1. Use the ECDSA algorithm with SHA-256.
4. Develop a verification function that takes the public key, the hashed message, and the generated signature as inputs and verifies the message's authenticity.

Recommended reading:
https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
https://cryptography.io/en/latest/hazmat/primitives/asymmetric/ec/

Example code format:

```
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import ec, utils
def digital_signature_EC():
#your code goes here
if __name__ == "__main__":
# write your code
```